

Experiment: Lego® Motor Characteristics

Aim

To experimentally determine the characteristics for the supplied geared Lego® motors for modelling the motors and controller design.

Equipment

- 1 ammeter
- 1 voltmeter
- 2 Lego® motors (as labelled), type 43362
 - Motor 0
 - Motor 1
- 1 ohmmeter
- 1 STK500
- 1 Computer
- 1 Oscilloscope
- 1 ATmega16
- 1 Variable lab power supply
- 1 known mass (about 100g)
 - A set of scales to measure mass
- 1 IR LED
- 1 IR phototransistor
- some Lego® beams
- 1 translucent black/white disc
- 1 breadboard
- 1 8.000 MHz crystal
- Vernier callipers

Procedure

1. Motor Armature Resistance

1. Take motor0 (marked by a '0' on one corner), and place the ohmmeter over the terminals
2. Record the resistance shown on the ohmmeter
3. Repeat for motor1

Results

Table 1

	Motor0	Motor1
Armature Resistance (Ohms)		

2. Motor Velocity Constant

1. Build a base for motor0 out of Lego® bricks, such as the one shown in Figure 1

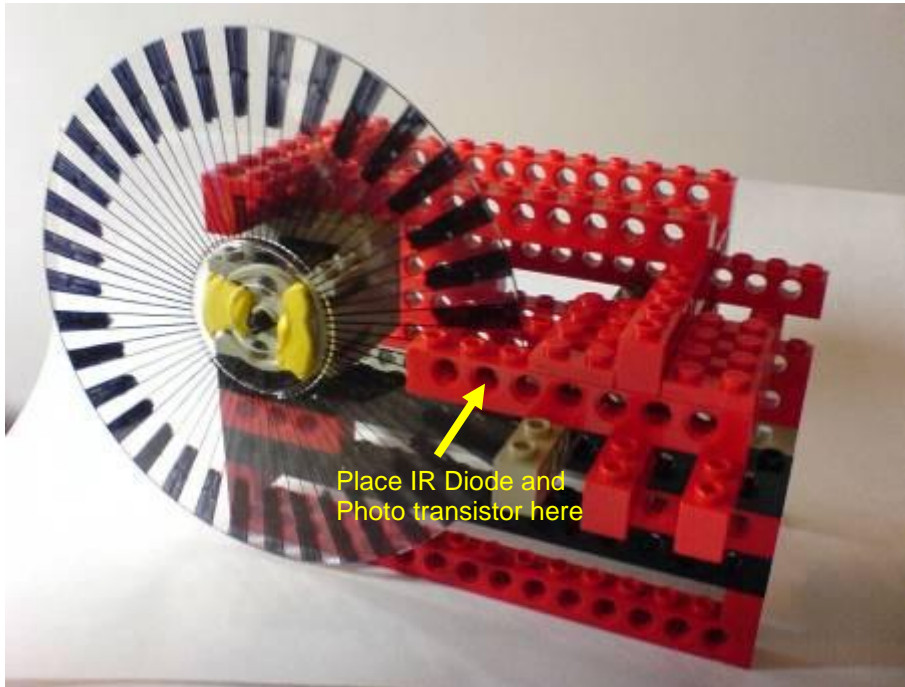


Figure 1

2. Attach a translucent disc with black/clear gradations on it
I have created a rotor with 5° spacing (72 gradations) on it as shown above in Figure 1.
3. Connect the circuit shown in Figure 2 on the breadboard.
Use R to control the current through your IR LED, use the datasheet to calculate a good current using Ohm's Law
4. Connect the breadboard to your STK500
5. Connect the motor to the lab power supply at 9V with an ammeter in series

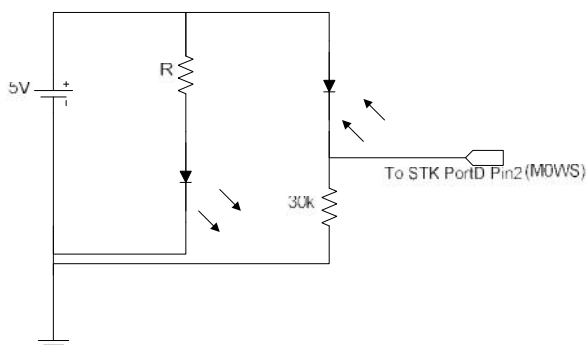


Figure 2

2.a Measure using a Microcontroller

6. Place an ATmega16 on your STK500
7. Place an 8.000 MHz crystal on your STK500
You will need to change jumpers on your STK500 to enable the crystal
8. Load the program from Appendix A onto the ATmega16.
9. Connect to the ATmega16 via STK500 RS232 'spare' port
10. Start hyperterminal, and use 57600 as the baud rate
11. Start the motor spinning by applying 9V, and wait for it to stop accelerating
12. Record the angular velocity displayed on the screen in Table 2
13. Record the current going through the motor in Table 2
14. Calculate K_N for the motor using the equation $K_N = \frac{i \cdot R_a}{\omega}$
15. Repeat for voltages 8, 7, 6, 5, 4, and 3 volts
16. Repeat for Motor1

2.b Measure using an Oscilloscope

6. Instead of connecting the breadboard to your STK500, connect the output to an input of your oscilloscope
7. Start the motor spinning by applying 9V, and wait for it to stop accelerating
8. Auto-set the oscilloscope
9. Measure the frequency of the input channel
10. Divide the frequency by 36 (72/2) and convert to the units $\text{rad} \cdot \text{s}^{-1}$ (hint: multiply by 2π).
11. Record the angular velocity calculated in 9. in Table 2
12. Record the current going through the motor in Table 2
13. Calculate K_N for the motor using the equation $K_N = \frac{i \cdot R_a}{\omega}$
14. Repeat for voltages 8, 7, 6, 5, 4, and 3 volts
15. Repeat for Motor1

Table 2

	Motor0	Motor1
Volts (1) ... 9V		
Angular Velocity (1)		
Current (1)		
Kn (1)		
Volts (2) ... 8V		
Angular Velocity (2)		
Current (2)		
Kn (2)		
Volts (3) ... 7V		
Angular Velocity (3)		
Current (3)		

	Motor0	Motor1
Kn (3)		
Volts (4) ... 6V		
Angular Velocity (4)		
Current (4)		
Kn (4)		
Volts (5) ... 5V		
Angular Velocity (5)		
Current (5)		
Kn (5)		
Volts (6) ... 4V		
Angular Velocity (6)		
Current (6)		
Kn (6)		
Volts (7) ... 3V		
Angular Velocity (7)		
Current (7)		
Kn (7)		

3. Motor Torque Constant

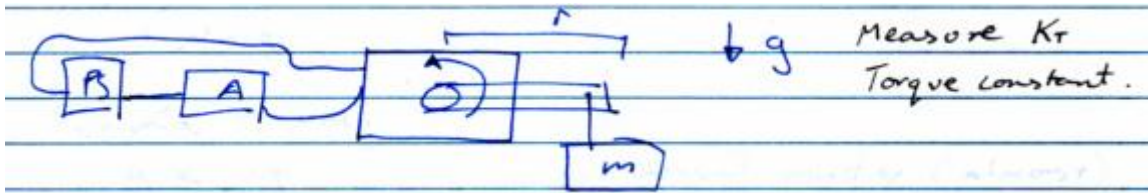


Figure 3

1. Build a base for motor0 out of Lego® bricks, such as the one in Figure 4
2. Attach a Lego® arm to the motor shaft as shown below in Figure 4

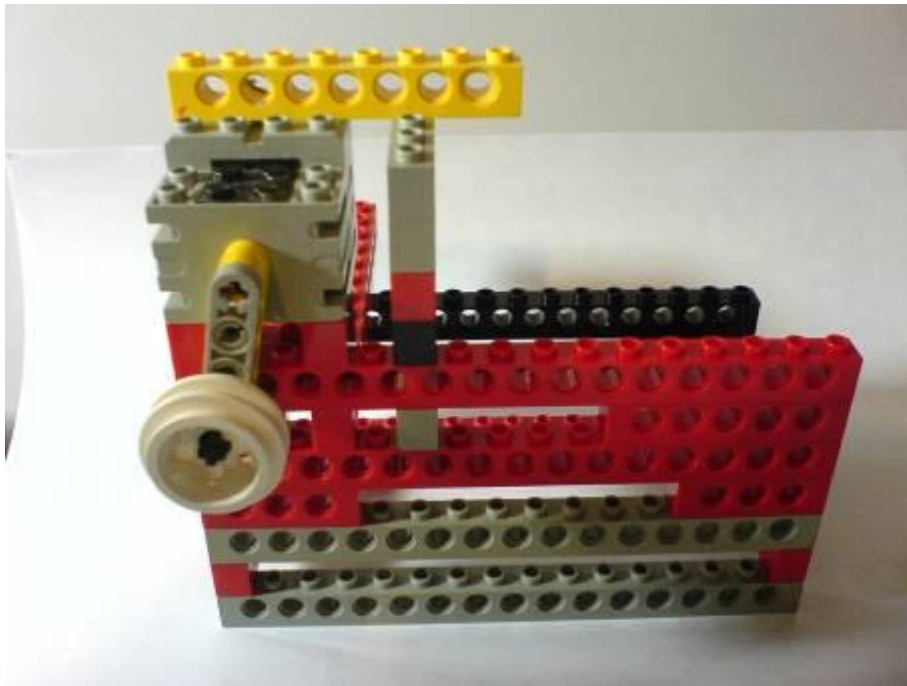


Figure 4

3. Record the arm radius from the motor shaft to the hanging position of the mass below in Table 3
4. Attach a known mass to the end of the arm.
5. Supply a known voltage (9V) to the motors with an ammeter in series as shown in Figure 3
6. Record the current shown on the ammeter in Table 3
7. Repeat steps 4 through 6 for several masses, suggested 10g, 50g, 100g, 150g, 200g
8. Repeat steps 1 through 7 for Motor1
9. Calculate torque from the equation $\tau = r \times 9.81m$ and record the results in Table 3
10. Calculate the torque constant from the equation $K_T = \frac{\tau}{i}$ and record the results in Table 3

Results

Table 3

	Motor0	Motor1
Arm radius		
Mass attached (g) 1		
Mass attached (g) 2		
Mass attached (g) 3		
Mass attached (g) 4		
Mass attached (g) 5		
Current (A) 1		
Current (A) 2		
Current (A) 3		
Current (A) 4		
Current (A) 5		
Torque (N.m) 1		
Torque (N.m) 2		
Torque (N.m) 3		
Torque (N.m) 4		
Torque (N.m) 5		
Torque Constant (N.m/A) 1		
Torque Constant (N.m/A) 2		
Torque Constant (N.m/A) 3		
Torque Constant (N.m/A) 4		
Torque Constant (N.m/A) 5		

4. Dead Zone Constant

1. Take motor0 and connect it to a variable lab power supply
2. Connect a voltmeter across the power supply
3. Set the variable lab power supply to 9V
4. Turn the power supply on
5. Turn down the voltage slowly until the motor stops moving
6. Record the voltage shown on the voltmeter in Table 4
7. Set the voltage to 0V
8. Turn up the voltage slowly until the motor just starts moving
9. Record the voltage on the voltmeter in Table 4
10. Average the stop and start voltages and record in Table 4
11. Repeat for Motor1

Table 4

	Motor0	Motor1
Voltage when stops (V)		
Voltage when starts (V)		
Average (V)		

5. Stall Current

1. Take motor0 and connect it to a 9V source
2. Connect an ammeter in series
3. Force the motor to stall (supply enough force such that nothing moves).
4. Record the current flowing through the motor in Table 5
5. Repeat for motor1

Table 5

	Motor0	Motor1
Stall Current (A)		

6. Evaluation

Evaluate the measurements by comparing the characteristics of each motor. Take K_n and see that it matches with the value of K_t for the motor.

Appendix A

```

/*
 * THIS PROGRAMME IS UNTESTED AND COMES WITH NO WARRANTIES ON OPERATION
 * COPYRIGHT 2006 DAVID L. SMITH
 * Written: 28 October 2006
 */

#include <avr/io.h>
#include <stdio.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>

#define SYSTEM_TIMER_INTERVAL 1
#define SYSTEM_TIMER_SCALE 256
#define TIMER1_INTERVAL 10
#define TIMER1_TIMER_SCALE 64
#define F_CPU 8000000UL
#define TCNT2_BOTTOM (unsigned char)(256 - (SYSTEM_TIMER_INTERVAL * 1000) *
(F_CPU/SYSTEM_TIMER_SCALE)/1000000)
#define TCNT1_BOTTOM (unsigned short)(65536 - (TIMER1_INTERVAL * 1000) *
(F_CPU/TIMER1_TIMER_SCALE)/1000000)

#define BAUD 57600L
// equation from Peter Sutton's example in COMP1300
#define UBRR_BAUD (unsigned short)((((F_CPU / (8 * BAUD)) + 1)/2 - 1))

volatile unsigned short angular_displacement;
// time between displacement interrupts
volatile unsigned short time;
volatile unsigned short ticks;
volatile unsigned char timer2_ticked_flag = 1;
volatile unsigned char timer1_ticked_flag = 1;

// setup a small buffer
// borrowed ring buffer code from Peter Sutton's example in COMP1300
#define SERIAL_TXD_BUFFER_SIZE 100
volatile static char serial_txd_buffer[SERIAL_TXD_BUFFER_SIZE];
volatile static unsigned char serial_txd_buffer_pos = 0;
volatile static unsigned char serial_txd_buffer_bytes = 0;

int serial_transmit(char data, FILE *stream)
{
    if (data == '\n') serial_transmit('\r', stream);
    loop_until_bit_is_set(UCSRA, UDRE);
    while (serial_txd_buffer_bytes >= SERIAL_TXD_BUFFER_SIZE);
    //UDR = data;
    serial_txd_buffer[serial_txd_buffer_pos++] = data;
    serial_txd_buffer_bytes++;
    // ring buffer
    if (serial_txd_buffer_pos == SERIAL_TXD_BUFFER_SIZE)
    {
        serial_txd_buffer_pos = 0;
    }
    UCSRB |= (1 << UDRIE);
    sei();
    return 0;
}

```

```

}

unsigned char serial_recieve(void)
{
    while(!(UCSRA & (1<<RXC)));
    return UDR;
}

static FILE mystdout = FDEV_SETUP_STREAM(serial_transmit, NULL, _FDEV_SETUP_WRITE);

float angular_velocity = 0;
volatile unsigned short ad_sample = 0;
volatile unsigned short time_sample = 0;

int main()
{
    // initialise timer2
    TCCR2 |= (1<<CS22) | (1<<CS21) | (0<<CS20);
    TIMSK |= (1<<TOIE2); // Timer 2 overflow interrupt enable
    TCNT2 = TCNT2_BOTTOM;

    // initialise timer1
    TCCR1B |= (0<<CS12) | (1<<CS11) | (1<<CS10);
    TIMSK |= (1<<TOIE1);
    TCNT1 = TCNT1_BOTTOM;

    // initialise external interrupt
    // INT0 initialise
    GICR |= (1 << INT0);
    // INT0 triggers on logic level change
    MCUCR |= (0 <<ISC01) | (1 << ISC00);

    // initialise serial
    UBRRH = (unsigned char)(UBRR_BAUD>>8);
    UBRRL = (unsigned char)(UBRR_BAUD);
    UCSRB = (1<<RXEN) | (1<<TXEN);
    UCSRC = (1<<URSEL) | (0<<USBS) | (3<<UCSZ0);
    stdout = &mystdout;

    unsigned char milliseconds;
    for (;;)
    {
        if (timer1_ticked_flag)
        {
            timer1_ticked_flag = 0;
            milliseconds += 10;
            // do the following every 100 ms (or 10 times a second)
            if (milliseconds == 100)
            {
                milliseconds = 0;
                angular_velocity = ad_sample * (5 / 180.0 * 3.142) /
(float)time_sample;
                ad_sample = 0;
                time_sample = 0;
                printf_P(PSTR("Time/Angular Displacement: %d/0.087 s/rad"),
time);
                printf_P(PSTR("\nTotal Angular Displacement: %d"),
angular_displacement);
            }
        }
    }
}

```

```

        printf_P(PSTR("\nTime averaged Angular Velocity (last 100
ms):"));
        printf_P(PSTR("\n\t%f rad/s"), angular_velocity);
    }
}
return 0;
}

ISR(USART_UDRE_vect)
{
    if (serial_txd_buffer_bytes > 0)
    {
        if(serial_txd_buffer_pos - serial_txd_buffer_bytes < 0)
        {
            UDR = serial_txd_buffer[serial_txd_buffer_pos -
serial_txd_buffer_bytes + SERIAL_TXD_BUFFER_SIZE];
        }
        else
        {
            UDR = serial_txd_buffer[serial_txd_buffer_pos -
serial_txd_buffer_bytes];
        }
        serial_txd_buffer_bytes--;
    }
    else
    {
        UCSRB &= ~(1<<UDRIE);
    }
}

// this timer interrupts every 1 ms
ISR(TIMER2_OVF_vect)
{
    sei();
    TCNT2 = TCNT2_BOTTOM;
    timer2_ticked_flag = 1;
    time++;
    ticks++;
    time_sample++;
}

// this timer interrupts every 10 ms
ISR(TIMER1_OVF_vect)
{
    sei();
    TCNT1 = TCNT1_BOTTOM;
    timer1_ticked_flag = 1;
}

// every time the level changes, increment the displacement
ISR(INT0_vect)
{
    angular_displacement++;
    ad_sample++;
    time = 0;
}

```